# Efficient Cover Song Identification using approximate nearest neighbors

Romain Tavenard
IRISA / University of Rennes I

Herv Jgou
INRIA Rennes

Mathieu Lagrange
IRCAM / CNRS

February 22, 2012

**Abstract**

Automatically detecting cover songs imply being robust to several kinds of musical modulations. Timbral variance can be accounted at the feature level, but key and most importantly tempo variations have to be dealt with at the retrieval stage. For that purpose, most state of the art approaches consider exhaustive search based on song to song matching methods that fail at scaling up.

In this paper, we introduce a hybrid technique. It first retrieves the approximate neighbors of each query chroma descriptor. In a second stage, the temporal consistency is exploited to further filter out some matches, thereby filtering irrelevant songs. Our method performs a search in a dataset comprising 80 songs in about 1s, while achieving satisfactory accuracy compared to the best performing techniques of the state of the art.

## 1 Introduction

Music retrieval is a wide problem. The typical tasks in the literature range from what is usually coined "music similarity", which typically consists in predicting a more or less loosely defined tag such as the musical genre [10]. At the other extreme, fingerprinting [2] aims at detecting the use of a specific audio recording. In between lies a class of alternative tasks, the most widely considered being the detection of cover songs. They distinguish themselves within each other depending on the kind of invariants that are elicited. Are we seeking for different performances of the same musical piece by the same performers, by different performers ? Shall the instrumentation be preserved ?

In this paper, we consider covers to be "different renditions of the same underlying musical piece", as previously used [6,15]. More precisely, two renditions are not constrained to be interpretations of the same musical score. Indeed, a

listener usually considers that a song is a cover of another one if they share some kind of harmonic progression, *i.e.*, chord progression and melody line. To reflect this, most algorithms focus on detecting longest sequences of "tonal content" that match within the two songs to be compared. The issue is then to be able to perform this detection in a reliable yet efficient manner within large scale databases.

This challenge is crucially linked to the notion of temporality which is arguably a key aspect in music. In music similarity search, the categories to be considered are so loosely defined that accounting for temporality is often neglected. In audio fingerprinting, the tempo and timing shall not vary. Therefore, short-term temporality is usually encoded for increasing precision.

When considering covers, the tempo and the timing vary from rendition to rendition. There are several way to address this issue. The first is to consider that the tempo is known as a prior knowledge. Assuming that the tempo can reliably be estimated and that the timing variations are negligible, one can compute tonal features that are tempo aligned [6]. The second is to design a matching algorithm that is robust to those changes, by allowing the warping of the time axis [15]. Such approaches lead to the highest accuracy so far, but the matching algorithm is not scalable due to his prohibitive complexity. Another class of methods build a temporal model for each song and evaluate the prediction capability of the model of a query given the tonal sequence of a potential match [14].

However, all those methods imply an exhaustive search of the database, which heavily constraints their scope of applicability. In [1], indexing technique based on inverted index search is considered. The approach is indeed scalable in time as the average query time is linear in function of the database size. However, the accuracy performance is significantly decreased as the database grows in size, most likely because they do not assume any sequentiality of the features, which increases the chance of false positive matches.

In this paper, we propose a method which is geared towards efficiency and allows us to propose a cover detection scheme that is able to deal with large scale databases. The algorithm is composed of two main contributions. First, a fast nearest neighbor strategy allows us to retrieve the most probable matches within the database. Secondly, incoherent matches are filtered out, with respect to some rules that encodes efficiently timing constraints that are robust to tempo changes. The paper is organized as follows: first, previous work is reviewed in Section 2 in order to motivate our approach that is introduced in Section 3. The proposed algorithm is then evaluated in terms of accuracy and scalability in Section 4. Those results are then discussed in Section 5.

## 2 Related Work

Cover song detection is an interesting methodological problem as it comes with well defined ground truth and good agreement on the musical concepts that should be encoded, allowing researchers to focus on the design of relevant re-

trieval techniques.

## 2.1 Tonal features

In general, one assumes that versions of the same piece preserve the main melodic line and/or the harmonic progression, regardless of its main key. Therefore, tonal or harmonic content is the most employed characteristic in version identification, by means of the so-called Pitch Class Profile (PCP) or chromas. Numerous variations have been presented in order to extract PCP features, and the reader is referred to [7] for more details on this topic.

## 2.2 Matching of tempo aligned chromagrams

It is proposed in [6] to make the matching robust to tempo change by assuming knowledge of the tempo (provided by a tempo and beat estimation algorithm). Each song is then represented as a sequence of chroma, each representing the tonal content of one beat. The matching score is then given by the maximal cross-correlation between the possibly transposed chromas sequences. Performance may be enhanced by tuning the pre-processing step, *e.g.* the beat tracker. Different biases for the average BPM were tested and it was found that 120 BPM was the best performing. As this bias may also influence the beat positions, several matching can be made if considering several BPM assumptions (120 and 240) for each song. Keeping the best one between the four possible correlations further improved the results, see Table 1.

## 2.3 Matching using dynamic programming

[15] suggest to evaluate similarity between two songs by means of their longest common subsequence, computed using dynamic programming. They start by transposing the query into the tonality of the candidate song by estimating the "Optimal Transposition Index", as described in [13].

The similarity estimation then consists in finding a path in a binary similarity matrix $S^k$, called "Cross Recurrence Plot" and defined as:

$$S^k_{i,j} = (q_i \in \mathrm{NN}_k(c_j, Q)) \wedge (c_j \in \mathrm{NN}_k(q_i, C)), \tag{1}$$

where $q_i$ (resp. $c_i$) is the $i$-th element of the query (resp. candidate) sequence and $\mathrm{NN}_k(c_j, Q)$ denotes the set of $k$-nearest neighbours of vector $c_j$ among the set of vectors in the query sequence $Q$.

In order to find flexible paths in this matrix, allowing for local edition operations such as insertions, deletions or replacements, the similarity is defined using a specifically tailored dynamic programming algorithm.

## 2.4 Modeling temporality

An alternative approach is to build a model of the temporal evolution of the features for each song. The prediction capability of this model given another

song indicate whether the two songs could match. Then, the cross-prediction error can be regarded as a measure of similarity of the two songs to be compared. The main difficulty resides in the choice of the temporal model. In [14], Thresholded Auto Regressive (TAR) models [16] are considered which consists of a collection of AR models where each single one is valid only in a certain domain of the reconstructed state space identified using a clustering technique.

### 2.5 Indexing approaches

In [1], a *Bag Of Features* (BOF) approach is considered to describe songs. First, features are quantized in order to encode the indices of their (ordered) 3 most energetic dimensions. Then, each song is arbitrarily segmented into chunks of equal length and each of these chunks is represented by the multiset of its quantized features. Quantization indices are the keys of an inverted file used to index the database. Then, at query time, cosine similarity between the multiset $M_{Q_i}$ of the $i$-th chunk of the query and the multiset $M_{C_j}$ of the $j$-th multiset of the database candidate is computed by querying the inverted file for each element in $M_{Q_i}$.

Finally, the similarity between sequences $Q$ and $C$ is computed as the product of cosine scores between segments $Q_i$ and their best match in $C$.

## 3 Proposed Approach

Our approach is based on an approximate (temporally) local $k$-nearest neighbour search and an adequate voting strategy. Temporal information is deliberately ignored at the first step so as to allow fast comparison, which efficiently produces the list of nearest neighbors for each descriptor of the query track. The voting strategy builds upon the results of this comparison to filter out temporally incoherent matches and ponderate scores.

Note that this process is repeated for each of the $d$ possible transposition indices for the query. In the following, we describe our algorithm for a given transposition index, the last step of our similarity evaluation being to retain the highest score obtain among all indices, corresponding to the best matching transposition.

### 3.1 Local nearest-neighbour approximation

To improve the efficiency of our scheme, we use an approximate nearest neighbor (ANN) search algorithm trading accuracy against efficiency. The search is performed in an indexing structure containing the set of vectors associated with *all* the database songs.

The most popular methods for this kind of search are Locality Sensitive Hashing [3] and the FLANN algorithm [11]. However these methods require to access the full vectors for post-verification, which limits the number of descriptors and therefore the bumber of songs that can be considered without accessing

disk storage. We, instead, use a recent method which was shown successful to index billions of vectors [9]. It is an extension of the product quantization method [8], which is based on inverted lists and quantization codes associated with the indexed vectors. The key parameters of this product quantization method, referred to as IVFADC in [8], are:

- The total number of inverted lists. It is set to 1024 in our case;
- The number $M$ of inverted lists visited per query descriptor. This parameter has the most impact on efficiency. We use $M$=4.
- The number of bytes per descriptor in the inverted file system, which is fixed to 6 in our case (+4 bytes for the descriptor identifier).

Although the approach of [8] returns mostly correct neighbors, the distance values obtained from the codes are not very accurate. We, therefore, use the vector re-ranking stage used in [9] to improve the distance estimation and improve the quality of the nearest neighbors. In essence, this re-ranking stage avoids using the raw chroma descriptors by encoding the error vector using quantization codes. This introduces an additional memory cost of 6 bytes per descriptor depending on the desired precision. This parameter has an important impact on memory, but little impact on efficiency, as the re-ranking is performed on a short-list of $2k$ neighbors only, where $k$ is the number of nearest neighbours we want to retrieve.

Overall, each descriptor is represented by $b = 16$ bytes. This is a crucial point that motivates the use of this approach in our system. The method used in this paper has successfully been used to index a billion-sized dataset without resorting to costly disk accesses. When transposed into a cover song detection framework, where sequences are in the order of 1,000 features long, this means that a million songs could be efficiently indexed this way, using around 20GB memory. In contrast, Locality Sensitive Hashing [3] requires a lot of hash functions to achieve good performance, and needs the raw descriptors to establish the final ranking.

On output of this stage, we have a list of nearest neighbors, which are then sorted by database song and query timestamp. For a given database song, we will denote in the following the matches by $\mathcal{L}_m = \{m_1, \ldots, m_n\}$ where $m_i = (t_{\mathrm{q}}^i, t_{\mathrm{c}}^i, s^i)$ is made of the timestamp $t_{\mathrm{q}}^i$ of the vector in the query that matches a vector at timestamp $t_{\mathrm{c}}^i$ in the candidate sequence with a score $s^i$. The matches in the list $\mathcal{L}_m$ are therefore ordered in ascending order of $t_{\mathrm{q}}^i$ and, if necessary, $t_{\mathrm{c}}^i$.

## 3.2 Voting strategy

Once a set of approximate $k$-nearest neighbours is computed for each vector of the query sequence, we compute a score for each match based on considerations of burstiness and then perform temporal robustification so as to filter out incoherent matches.

**Handling Multiple matches.** For a given vector of the query track, it often occurs that several vectors from the same database song are returned. These vectors correspond to different timestamps in the database song. These vectors have therefore a higher contribution than what would be desirable in the cover similarity measure based on nearest neigbhors. In order to limit their contribution, if multiple matches of the same database are associated with the same query vector, then their vote score is downweighted. This is done by choosing a voting score of 1 divided by the square root of the number of matches of the same song that have voted for this particular query vector. For instance, if two vectors of the database vector are nearest neighbors of the query vector, then their voting contribution is $1/\sqrt{2} \approx 0.707$ instead of 1 for an isolated match.

**Temporal robustification.** Let us consider a candidate sequence $C$ that shares some matches with the query sequence. The aim is then to select the largest set of consistent matches in $\mathcal{L}_m$. A pair $(m_i, m_j)$ of matches is consistent if and only if it satisfies the following two inequalities:

$$(t_{\mathrm{q}}^i - t_{\mathrm{q}}^j) \times (t_{\mathrm{c}}^i - t_{\mathrm{c}}^j) > 0 \quad (2)$$

$$\left| \frac{t_{\mathrm{q}}^i - t_{\mathrm{q}}^j}{t_{\mathrm{c}}^i - t_{\mathrm{c}}^j} \right| \in \left[ \frac{1}{r^*}; r^* \right] \quad (3)$$

These inequations[1] ensure that timestamps of $i$ and $j$ are coherent in the sense that if the query timestamp in $i$ is lower (resp. greater) than the one in $j$, the candidate timestamp has to be lower (resp. greater) in $i$ than in $j$ too. Moreover, the parameter $r^*$ controls the expected maximum beat ratio between two songs. A set of matches is said consistent if any pair of matches in this set satisfies this consistency check.

In order to assess similarity between sequences, we compute the maximum total score of all sets of consistent matches in $\mathcal{L}_m$. Algorithm 1 presents a way to compute such a score while only maintaining a table of scores of length $n$, the number of matches.

Figure 1 illustrates the set of matches found by this algorithm for two particular query/database tuples. Here, the song is a cover of the query, therefore the total number of matches is relatively high. The line corresponds to the path connecting the matches that have been selected as belonging to the best set of consistent matches. Note the small proportion of matches that are temporally consistent, which justifies the use of such a filtering step.

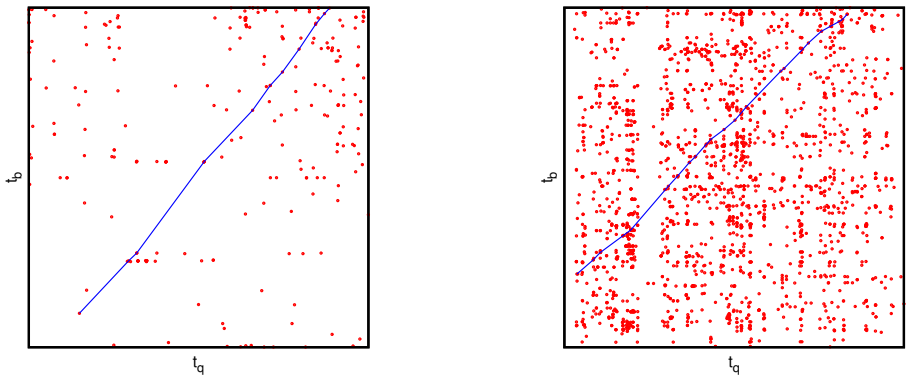
**Score normalization.** The score obtained by the voting scheme is not regularized. Songs with many descriptors are, therefore, favored compared to those with fewer descriptors. Different normalization schemes have been evaluated [12] in the context of image retrieval, some of which are applicable in a BOF framework only, such as the L2 normalization of the BOF vector. In our voting framework, we divide the score by the square root of the number of descriptors,

[1] In practice, checking if these criteria are satisfied is done based on a single test, by removing the absolute value in the second inequality.

**Algorithm 1** Scoring algorithm that sums consistent matches together.

**Require:** $\mathcal{L}_m = \{m_1, \ldots, m_n\}$, $r^*$

```
bestsofar ← zeros (n)
for i = 1:n do
    for j = (i + 1):n do
        if (m_i, m_j) is consistent w.r.t. r* then
            score ← bestsofar[i] + s^i
            bestsofar[j] ← max(bestsofar[j], score)
        end if
    end for
    bestsofar[i] ← bestsofar[i] + s^i
end for
return max(bestsofar)
```




(a) Cover song (b) Non-cover song

Figure 1: Temporal consistency. On input, our algorithm takes a set of matches retrieved by approximate search (displayed by points). On output, it produces the best subset of matches that are consistent, here represented by a line connecting the selected matches. The same query is considered for both figures. On the left, the database song is a cover of the query song, while it is not the case for the right-hand side figure. Note than even though the candidate song in (b) has much more matches, it is ranked after the one of (a) due to both our anti-burstiness strategy and our temporal consistency checking.

*i.e.* the normalized similarity $s(a,b)$ is obtained as:

$$s(a,b) = \frac{s(a,b)}{\sqrt{n_a}\sqrt{n_b}}, \tag{4}$$

where $n_a$ and $n_b$ correspond to the number of descriptors in song a and b, respectively.

This square root normalization has the desirable property that $s(a,b) = 1$ if each descriptor of $a$ is matched with itself only, in which case $s(a,a) = n_a$. Note that this property is also satisfied in a bag-of-word framework if the cosine similarity is used.

**The complexity** of this last stage may appear costly, as it includes dynamic programming and is therefore quadratic in the number of matches obtained for a particular query/base tuple. However, in practice this number of matches is very small for the vast majority of database elements. This is because the total number of nearest neighbors retrieved by the ANN algorithm is $k \times n_{\mathrm{q}}$, where $n_{\mathrm{q}}$ is the number of descriptor in the query. Only a limited fraction of these neighbors are associated with a particular database song. In the worst case, all the neighbors would be associated with the same song[2], leading to a total complexity of $k^2 \times n_{\mathrm{q}}^2$ to compare the query with *all* the database elements.

# 4 Experiments

## 4.1 Datasets

We evaluate our algorithm on the well-known **Covers80** dataset [5]. This dataset is made of 80 pairs of pop songs, each pair being made of two different versions of a same song. All covers are performed by different artists from the original versions. We used the software of [4] to extract 12-dimensional chroma features, using a hop size of 125ms and an analysis window size of 250ms.

## 4.2 Evaluation protocol

The search quality is measured as the recall@$k$, *i.e.*, the proportion of queries whose nearest neighbor is ranked in the first $k$ positions. Also, the recall@$k$ is the fraction of queries for which the nearest neighbor would be retrieved correctly if a short-list of $k$ songs was verified using a perfect re-ranking strategy. The efficiency is measured by actual timings.

## 4.3 Evaluation of the proposed approach

In the following, we use, for our algorithm, a coarse quantizer dividing the feature space into 1024 cells and explore, for each query vector, the 4 closest cells. Doing so, we compute approximate distances for $1/256^{\mathrm{th}}$ of the database. As stated above, each feature from the database is encoded using 16 bytes.

[2] In practice, this is obviously not the case

Figure 2: Influence of the maximum tempo ratio on the quality of results.

| Method | Variant | recall@10 |
|---|---|---|
| Our approach | $k = 5$ | 0.463 |
| | $k = 20$ | 0.513 |
| | $k = 50$ | 0.500 |
| LabROSA | Baseline | 0.513 |
| | Tempo bias = 120 BPM | 0.600 |
| | Dual tempo levels | 0.675 |

Table 1: Results obtained on the covers80 dataset. The varying parameter is the number of nearest neighbours considered for the vote.

We study in the following the influence of the number $k$ of nearest neighbors retrieved and the tempo ratio $r^*$. When not explicited, the former is set to 20 and the latter to 1.4.

We compare our approach to the results obtained by the 2007 LabROSA cover detection system [6] on the covers80 dataset, so as to assess its quality. We also report timings showing the efficiency of our algorithm on this dataset.

Results obtained for are presented in Table 1 and show that using only the first few nearest neighbours for each query vector performs better, which allows cheap (in terms of time consumption) temporal robustification, as the latter is a function of the number of matches (see Section 3.2 for more details). Note however that tuning parameters on such a small dataset is difficult as the results can be noisy due to the small number of songs to retrieve. Moreover, when considering a larger database, as votes are distributed among a higher number of songs, it is expected that scores would be lower, implying the need for larger values of $k$ so as to perform a more appropriate consistency check.

We obtain comparable results to the baseline of LabROSA, which is positive, as our algorithm enables scaling up without comparing a query song with every candidate song in the database. Also, our algorithm only considers little temporal information: the only prior knowledge is that the ratio of tempo between two versions of a song $r^*$ should be in the range $[1/r^*; r^*]$.

Figure 2 shows that setting reasonable values for $r^*$ has little impact on the quality of retrieval, while an optimum is reached for $r^* = 1.5$.

Table 2 also reports timings for the same values of $k$, divided into the time spent for the approximate nearest neighbour search and the one for the voting scheme. Notice that the latter is always much lower than the former, which confirms the statement made in Section 3.2 that its complexity is not a limiting factor[3].

[3]Note that these timings are reported when running our experiments on a single core, while our code is parallelized and can achieve much lower timings when run on several cores.

| Variant | Time per query for ANN (s) | Time per query for vote (s) |
|---|---|---|
| $k = 5$ | 4.09 | 0.30 |
| $k = 20$ | 5.06 | 3.85 |
| $k = 50$ | 6.46 | 20.92 |

Table 2: Timings to compare a query with *all* the 80 covers of Covers-80, averaged over the 80 queries and measured on 1 core. The timings of the first column depend on the database size (almost independent from $k$). The time consistency check (second column) mainly depends on $k$. Note that using our multi-threaded implementation (12 cores, 1 per transposition), we obtain query times of about 1 second for $k = 20$.

# 5 Discussion

In this paper we introduced a new approach for cover song identification. It is based on a local estimation of nearest neighbors and a temporal consistency check tailored to filter out matches that do not correspond to a realistic alignment of song segments. The results obtained on a small dataset are encouraging as the achieved search quality is comparable to already published results on the same dataset, at a very low computing cost. We aim at testing our algorithm on the recently published Second Hand Dataset, on which the large-scale capabilities of our algorithm could be enlightened.

# References


[1] E. Buccio, N. Montecchio, and N. Orio. A Scalable Cover Identification Engine. In *ACM International Conference on Multimedia*, pages 1–4, 2010.

[2] P. Cano, E. Battle, T. Kalker, and J. Haitsma. A Review of Audio Fingerprinting. *The Journal of VLSI Signal Processing*, 41(3):271–284, 2005.

[3] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Symposium on Computational Geometry*, pages 253–262, 2004.

[4] D. Ellis and G. Poliner. Identifying "cover songs" with chroma features and dynamic programming beat tracking. In *ICASSP*, pages 1429–1432, 2007.

[5] D. P. W. Ellis. The "covers80" cover song data set. web resource, available: http://labrosa.ee.columbia.edu/projects/coversongs/covers80/, 2007.

[6] D. P. W. Ellis and C. Cotton. The 2007 LabROSA cover song detection system. *MIREX extended abstract*, September 2007.

[7] E. Gómez. *Tonal description of music audio signals*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Spain, 2006.

[8] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *PAMI*, 33(1):117–128, January 2011.

[9] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: re-rank with source coding. In *ICASSP*, 2011.

[10] B. Logan and A. Salomon. A Music Similarity Function Based on Signal Analysis. In *ICME 2001*, pages 2–5. Citeseer, 2001.

[11] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, 2009.

[12] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *CVPR*, 2008.

[13] J. Serra, E. Gomez, P. Herrera, and X. Serra. Chroma binary similarity and local alignment applied to cover song identification. *IEEE Transactions on Audio, Speech, and Language Processing*, 16:1138–1151, 2008.

[14] J. Serrà, H. Kantz, and R. G. Andrzejak. Model-based cover song detection via threshold autoregressive forecast. In *In Proc. of the ACM Multimedia, Workshop on Music and Machine Learning (MML)*, pages 13 – 16, 2010.

[15] J. Serrà, X. Serra, and R. G. Andrzejak. Cross recurrence quantification for cover song identification. *New Journal of Physics*, 11:093017, September 2009.

[16] H. Tong and K. S. Lim. Threshold autoregression, limit cycles and cyclical data. *Journal of the Royal Statistical Society*, 42(3):245 – 292, 1980.